Towards a Cradle-to-Grave, Mission-Wide Simulation System

P. F. Maldague[1], S.S. Wissler[2]
Jet Propulsion Laboratory, NASA/California Institute of Technology, Pasadena, CA, 91109, United States

We articulate a vision for a multi-mission simulation framework capable of servicing a Space Mission over its entire lifetime, from early formulation to end-of-mission phases. Our vision adopts and extends the APGen-based simulation methodology used by NASA's planned Europa Clipper mission to examine key spacecraft trades, assess impacts to operability, and quantify how well the scientific objectives of the mission can be achieved. The simulation framework we propose will provide Europa Clipper and future missions with unprecedented high-fidelity, integrated, system-level capabilities with the potential to dramatically improve design, performance, science return and operability.

I. Nomenclature

AAF	= APGen Adaptation File
ACS	= Attitude Control System
APGen	= Activity Plan Generator
API	= Application Programming Interface
APID	= APplication IDentification
C&DH	= Command and Data Handling
CORBA	= Common Object Request Broker Architecture
DSN	= Deep Space Network
DSL	= Domain-Specific Language
ESA	= European Space Agency
FOV	= Field Of View
GNC	= Guidance, Navigation and Control
HTTP	= HyperText Transfer Protocol
IDE	= Integrated Development Environment
IMCE	= Integrated Model-Centric Engineering
ISO	= International Standards Organization
JAXA	= Japan Aerospace Exploration Agency
JPL	= Jet Propulsion Laboratory
JSON	= JavaScript Object Notation
MER	= Mars Exploration Rovers
MMPAT	= Multi-Mission Power Analysis Tool
MOS	= Mission Operations System
MSL	= Mars Science Laboratory
NAIF	= Navigation and Ancillary Information Facility
NASA	= National Aeronautics and Space Administration
OSI	= Open Systems Interconnection
OTM	= Orbit Transfer Maneuver
PEL	= Power Equipment List
RAP	= Reference Activity Plan
ReST	= Representational State Transfer
SEQGEN	= Sequence Generator
SOAP	= Simple Object Access Protocol
SOC	= State Of Charge
TCM	= Trajectory Correction Maneuver
TFP	= Telecommunications Forecast Predictor

[1] Engineering Applications Software Engineer, Mission Systems and Operation Division
[2] Chief Engineer, Mission Systems and Operation Division

UDEF	= User-DEFined library
XmlRpc	= eXtensible Markup Language Remote Procedure Call protocol

II. Introduction

The high-fidelity simulations of the planned Europa Mission described elsewhere in these proceedings [1] have demonstrated the enormous value of carrying out such simulations early in the life of a project in many areas:

- Mission Plan strategy optimization
- Trade studies involving spacecraft configuration, additional instruments, radiator position
- Allocations of energy and data
- Hardware design
- Hardware test plan development
- Operability
- Requirements verification
- Fault sensitivity analysis
- Reusable components for a potential Europa Lander and other future missions

In view of these achievements, it is natural to ask two questions about the future of the Europa simulation framework:

1. How should the framework evolve to support the Europa Mission beyond Phase B?
2. Can the framework be extended to other space missions?

The purpose of this paper is to provide answers to both questions. Although the answers will reflect the experience acquired by the authors in the development and use of the APGen-based simulation framework, they will nonetheless contain speculative elements regarding future development of the current simulation methodology. In particular, the identification of key future challenges involves human elements as well as technical issues, and some guesses will have to take place. Having identified the likely challenges, we will go on to try and predict the best approach to meet those challenges - which will require some more guesswork. As a result, the answers provided here have a certain aura of uncertainty about them. But one thing is certain: the current infrastructure needs to change if it is to support new customers. The predictions and recommendations found here may not be perfectly accurate, but they have the merit of addressing the problem. It is the authors' hope that, imperfect as they may be, these recommendations will provide useful guidance to space mission systems engineers in their quest for the best possible simulation system.

III. The Current Simulation Infrastructure

A necessary step towards providing guidelines for how to change a simulation infrastructure is to take a good look at the current state of that infrastructure with an eye towards improvement opportunities. To a large extent, the APGen-based simulation infrastructure grew from the bottom up. Over a period of about twenty years, the authors and their co-workers took advantage of existing modeling capabilities, extended these capabilities incrementally so as to avoid major and costly redesign, and used creativity and ingenuity whenever available capabilities did not quite measure up to the mission-specific task at hand. In this section, we review the results of this bottom-up evolution, which has led to the modeling infrastructure currently used by Europa Clipper and other space missions.

A. Technology is not the Issue

Interestingly, the technology required to perform high-fidelity mission simulations is not new. Discrete event simulation has been around for over twenty years. Technologies for gluing together applications and libraries that were not intended to work as an integrated system have been available for about as long: software components such as dynamic libraries and run-time loaders, communications protocols such as CORBA, SOAP and XmlRpc, and easy-to-parse file formats such as comma-separated values, XML and JSON have all been available for a number of years. Given that the technology required for performing high-fidelity simulations was available, and given the fact that such simulations are clearly very beneficial to the project, why did it take so long for simulations such as reported in Ref. [1] to become available to space missions? This turns out to be a complex question with several facets. We would like to offer answers in the form of two real-life lessons, learned the hard way:

> Lesson 1: it is easy to learn and apply design principles that combine low cost and high probability of success, but it is much more difficult to find guidance in how to integrate systems that were not designed to work together

Lesson 2: system-wide simulations of a complete space mission require an extremely broad range of domain-specific expertise and experience

It is possible to take lesson 1 into account by designing a simulation system from scratch, but then, lesson 2 guarantees that the cost will be very high. Over the years, the authors have witnessed a number of attempts to reduce cost by taking advantage of existing capabilities in designing a mission-wide system. What invariably happens is that lesson 1 ends up driving costs up no matter what, preventing the project from delivering its promise within the budget allocated to it.

B. APGen: from Planning Tool to Simulation Engine

The central tool used to build and run simulations of the Europa Clipper Mission in its current stage is APGen, a modeling and simulation framework developed by NASA's Advanced Multi-Mission Operating System (AMMOS) organization [2], [3]. The focus of the original requirements on APGen was mission operations, and in particular the planning function within the Mission Operations System (MOS). The purpose of APGen, as stated in the requirements, was to help mission planners create activity plans that did not oversubscribe critical resources such as data storage, electrical power and fuel during mission operations. Early users of APGen had to perform a fair amount of manual work when using APGen, such as dropping new activities into mission plans. Every now and then, the user would ask APGen to "model" the plan, i. e., to exercise the modeling rules encoded in the adaptation and evaluate the impact of the current plan on critical resources. Modeling was a time-consuming process, and APGen did not try to keep the resource model in synch with the changing activity plan unless requested to do so by the user.

Later, the capabilities of APGen evolved in response to evolving requirements from its users. This evolution is described elsewhere and will not be duplicated here [3]. What we will do instead is list the capabilities that APGen ended up with. Although these capabilities grew largely as a result of requests coming from missions in Phases D and E, they turned out to be extremely useful in the less well-defined environment of pre-Phase A through Phase C.

C. The APGen DSL or "Adaptation Language"

The first capability of APGen that turned out to be of crucial importance later on is its Domain Specific Language (DSL). The DSL is what mission engineers use to specify ground and spacecraft activities, resources, and their interaction. The DSL is tailored to increase code clarity and coding efficiency when compared to conventional languages. The fundamental constructs within the DSL are parametrized activity types (much like classes in object-oriented programs) that can be instantiated within a plan and resources which collectively describe the full state of the simulation. DSL code defining activities and resources specific to a particular mission is often referred to as an APGen "adaptation". Activities and their effect on resources are modeled using APGen's discrete-event simulation engine, which only performs calculations when requested to. Modeling requests originate either from the APGen user interface or, more frequently, from execution scripts written by the simulation developer to orchestrate the overall simulation process. The discrete event simulation paradigm allows APGen to run simulations significantly faster than real-time; for example, Europa Clipper end-to-end simulations run at over 8,000x real-time.

1. Strong but Flexible Typing of APGen variables

One feature of the DSL which is widely appreciated by APGen adapters is the flexibility it provides when defining simulation variables. The DSL is strongly typed in the sense that the type of each variable must be specified at the time the variable is defined. The following types are supported:

1. Boolean
2. integer
3. double-precision
4. string
5. time
6. duration
7. object
8. array

Automatic conversion between types is allowed in specific cases (e. g. between integer and double-precision values) but most conversions are forbidden. For example, a duration can be added to another duration or to a time, but not to an integer nor to a double-precision number (early versions of APGen allowed this, interpreting the number as a number of seconds.) In that sense, the APGen DSL is a strongly typed programming language.

Within an array, the style of the array (list or map) and the types of the array elements do not have to be declared until the elements are actually assigned specific values. An APGen array can have one of two styles: a list of values or a map of values in which each element is indexed by a string. In either case, the values do not have to be all of the same type. The actual type of a value is determined at the time an element is defined. For example, if a variable *A* has been declared as an array, an assignment in the style

A[0] = 3.141593;

will specialize the type of *A* as a list, while an assignment in the style

A["high gain antenna"]=2.0;

will specialize the type of *A* as a map. Once an array has been identified as a list (resp. map), additional element definitions must always use the first (resp. second) style. Furthermore, once an array element has been defined through an assignment in either style, the element inherits the type of the value assigned to it, which in effect becomes the declared type of the array element. Further assignments to that same element can only be made if the assigned value is of the same type as the element, or if APGen provides an implicit conversion from the type of the value to the type of the element.

2. Time and Duration as Native Types

We make a special note of the availability of time and duration types. Standard programming languages provide time specifications, such as the *time_t* type available from the *C* library, as well as utility functions for manipulating time values. The APGen DSL, in recognition of the all-important role played by time and duration in modeling algorithms, provides concise encapsulation of the most frequent operations related to times:

- durations can be added to or subtracted from both times and durations
- negative durations are allowed
- times can be subtracted, resulting in a duration
- a duration can be divided by a duration, resulting in a double-precision number
- a duration can be multiplied by a number, resulting in another duration

Time and duration values are implemented as fixed-precision quantities with a one-millisecond resolution. The reason for this is that time values are used as the primary index in the all-important queue of modeling events, and it would be extremely awkward to have to deal with rounding errors in that primary index. Furthermore, the timing algorithms in APGen are designed to be completely deterministic - modeling the same problem twice produces exactly the same result - and allowing rounding errors into these algorithms would conflict with the design.

3. Referencing Objects

Finally, we note that the object type is essentially the same as a “smart pointer” in languages such as C++. Currently, this type is only used to capture activity instances created by APGen's scheduling and decomposition algorithms. A more general implementation, allowing object variables to point to other objects such as resources and events, is currently in preparation. The purpose of this generalization would be to provide support for modeling disruptive events such as faults. When modeling an unplanned event, adapters need to establish and manipulate the context in which the event occurred, such as lists of activities whose execution was still in process at the time of the event. A general object type provides exactly the kind of access an adapter needs in such situations.

4. APGen DSL vs. Established Programming Languages

It is clear from the above that the APGen DSL differs in essential ways from generic programming languages such as C and python. As a result, learning to adapt APGen is a skill that presents a significant learning curve. Once the DSL has been mastered, its use allows adapters to express and even reuse their modeling patterns with considerable ease. But there is no question that there is a tradeoff between the learning curve and the corresponding benefits.

Fortunately, the alternative between the APGen DSL and more widely used programming languages does not have to be an either-or situation. As will be explored in detail later in this article, the APGen framework can be extended to provide external models with direct access to the modeling process, resulting in a distributed, heterogeneous modeling system in which a variety of model implementations can coexist peacefully. We will return to this topic in a later section of the paper.

D. Integration of External Models: the APGen User-Defined Library

Next, we mention another essential component of the APGen infrastructure: the user-defined library (UDEF). This library allows adapters to extend the DSL by providing new functions. The new functions must be implemented in a shared library which is loaded at run time by the APGen program. The shared library features a well-defined API

which tells adapters how to extract function arguments from the internal APGen data repository and how to define function results which can be returned to the APGen core engine for further processing.
The user-defined library has turned out to play an essential role in the ability to integrate APGen with an external model in case the external model is available as a C or C++ library. The user-defined library can be built in such a way that the external library is linked to it, either statically (as a static archive) or dynamically (as a shared library loaded at run time.) By including external library header files in the build process, the user-defined library API can make calls to external functions, thus exposing the functionality of the external library to the APGen DSL in a seamless manner. Although this integration mechanism is not intuitive and takes some getting used to, especially for non-programmers, it has proven very flexible and reliable over the years.

E. Hierarchical Activity Decomposition

A less-than-obvious but extremely useful feature of APGen is that it provides adapters with the ability to create and maintain hierarchies of activities which cooperate in achieving a desired state (a goal, in the terminology we will introduce later on) or outcome. The APGen DSL provides adapters with means to specify how a high-level activity, for example a maneuver, decomposes into child activities each one of which is responsible for one aspect of the overall purpose of the high-level activity. Decomposition algorithms allow high-level activities, which typically represent science or engineering goals, to be expanded recursively into lower-level activities and ultimately into commands that implement these goals. Early APGen activity plans contained a few hundred activities. Today, detailed multi-year simulations like those produced by the planned Europa Clipper mission produce between 100,000 and a million activities. The hierarchy mechanism is essential in providing means to organize such large activity plans into a structure that can be examined relatively easily.

F. APGen Scheduling Algorithms

APGen's scheduling capabilities turn out to be an essential part - perhaps even the most important part - of providing a fully functional simulation of a space mission. The basic scheduling algorithm used by the APGen modeling engine has been described in detail elsewhere [2], and we will therefore limit ourselves to a brief overview. The basic component of the APGen scheduling engine is a window finder which takes as input a “scheduling condition” and a minimum duration over which the scheduling condition is to be continuously true. The scheduling condition can be a generic expression of arbitrary complexity which evaluates to a Boolean value. The expression can involve constants, arithmetic and logical operators, calls to AAF-defined functions and special APGen methods (*currentval*) that return the current value of any state variable. The restriction to AAF-defined functions is motivated by the need to identify all events that can potentially cause the scheduling condition to become true.

An important restriction on the use of the window-finding algorithm is that it needs to base its evaluation of state variables on a previous modeling pass. Specifically, suppose that the activity plan is currently in a state which we denote symbolically by *A*. Suppose also that we have asked APGen to model the plan while in that state, so that the time-dependent profiles of all state variables reflect state *A* of the activity plan. Our goal is to find windows within which to schedule additional activities. Before any of these additional activities are created, the plan is still in state *A*, and we can rely on the previously modeled state variables to evaluate windows. Once we start adding activities within those windows, the plan enters new states *B, C, ...* as we keep adding activities to the plan one by one. Because modeling the entire plan is typically a time-consuming task, the scheduling engine only recomputes state variables up to the current value of simulation time (designated by the global variable *now* in the APGen adaptation.) State variable values beyond *now* are still provided by the modeling run that was performed earlier, while the plan was in state *A*. This restriction is not intuitive, and it takes time for an adapter to write algorithms that are robust in spite of it. On the positive side, the restriction makes scheduling about as efficient as modeling. Eliminating the restriction would require taking snapshots of the plan state before placing any new activity in the plan and restoring that state in case the new activity turned out to violate some constraint not included in the scheduling condition. Backtracking is of course a well-known technique in automatic activity planning; it is just not practical to use it given the computational complexity of typical APGen models.

In early applications, scheduling could only be performed on the fly, i. e., concurrently with the modeling process. In a recent development motivated by the needs of the Europa Clipper adapters, an asynchronous *get_windows* function was introduced to provide the window-finding capability outside of the modeling loop. The semantics of *get_windows* is simpler than the previous scheduling semantics, since the computation of windows has been decoupled from the modeling process. As a result, this new function has made scheduling easier to learn and easier to implement by APGen adapters.

IV. The Current Adaptation Process

In this section, we will outline the process that is used by mission personnel to customize, or "adapt," APGen to a specific mission such as Europa Clipper. The adaptation process described below is specific to the APGen tool. While this particular process has served Europa Clipper very well until now, we will argue later in our paper that future development will require taking steps away from the APGen-specific adaptation process and towards a more generic, system-wide process to support later Mission Phases.

In making recommendations for a generic, system-wide process to replace the APGen-centric system used up to now, it is very important to ensure that the essence of the current system has been captured correctly. We believe that the success of the APGen-based adaptation effort is due in large part to qualities and other "-ilities" that have not received the attention they deserve; in particular, we are thinking of things listed in Table 1 below.

Table 1 - Desirable "-ilities" of an Ideal Modeling System

-ility	Resulting requirement(s)
tweakability	The system can be easily modified to correct errors or omissions
evolvability	The system can be adapted or extended to add additional detail or new models with little effort
integrability	The components of the system are heterogeneous, yet they are able to cooperate in harmony thanks to interfaces that are easy to implement, configure and deploy
verifiability	The behavior of the system can be analyzed and validated without difficulty

A key concern as we navigate through the various subsystems that make up the APGen adaptation will be to capture the above "-ilities" as faithfully as we can, which will enable us to list them as requirements for the Ideal Simulation System later in our paper.

A. Adaptation Files

Besides the modeling engine itself, an APGen-based simulation must include one or, more typically, a number of files describing the components of the system and the precise manner in which these components interact. The process of assembling, coordinating and maintaining these files is known as "adaptation." The adaptation of a complex mission such as Europa Clipper involves one to two hundred APGen adaptation files (AAFs) and the adaptation task is therefore a significant part of setting up a complete simulation system.

To make it easier to manage the adaptation, it is customary to divide it into chunks known as "subsystems." A typical subsystem involves a dozen or so adaptation files, each one devoted to one aspect of the subsystem such as

- constants
- global variables
- lists of subsystem components
- lists of state variables (APGen resources)
- high-level models of subsystem behavior (APGen abstract resources)
- subsystem-specific activity types
- activity decomposition algorithms
- activity behavior specification

Table 2 below, which is adapted from Ref. [1], shows a list of the subsystems used on the Europa Clipper adaptation.

Table 2 - Europa Clipper Subsystems (adapted from Ref. [1])

Subsystem	***Description***
Geometry	NASA's Navigation and Ancillary Information Facility (NAIF) SPICE based geometry model
Ground Station	Model of Earth-based ground stations composed primarily of the Deep Space Network (DSN) plus assets from the European Space Agency (ESA), the Japan Aerospace Exploration Agency (JAXA) and NASA's Near Earth Network

Subsystem	***Description***
Telecommunications	Supports downlink modeling such as achievable data rates for X and Ka-band transmitters for all low-gain, fan-beam, medium-gain and high-gain antennas
Data	Model of instrument and engineering data production, on-board data storage, and playback of data to the DSN
Power	Models power loads from all spacecraft subsystems and instruments, solar array output and battery state of charge
GNC	Models commanded attitudes for communication, science observations, and trajectory maneuvers
Solar Array	Models solar array articulation for sun tracking and fixed modes as well as hard-stop avoidance "flops"
Radiation	Models the radiation environment's effects on the solar arrays
Propulsion	Medium fidelity fuel usage model
Payload	Instrument models based on specifications provided by the Europa science team
Mission Operations	Models mission operations process, timelines and shift schedules

B. Subsystem Organization

A typical subsystem contains a number of entities: global constants and variables, state variables or resources, activity types, constraints, and a variety of algorithms which describe things like how activities decompose into lower-level activities and how activities affect state variables and resources. The APGen DSL offers descriptive names for all these entities, as part of a language syntax which still contains many elements of its original design in 1996. In recent years, however, the Integrated Model-Centric Engineering (IMCE) initiative at JPL has embarked on a systematic attempt to capture the common elements of the many models used in the design, development and operation of space missions and to provide a uniform terminology for describing them [4]. Although IMCE is still a work in progress, it has the potential of making integration of heterogeneous models much easier than it is today, and we wholeheartedly support its efforts in that direction. As a modest contribution to IMCE's efforts, we have made an attempt to bridge the gap between the APGen DSL and what IMCE calls the *ontology* of a mission system. Table 3 below is a by-product of our effort; it lists the key elements of a generic subsystem in terms borrowed from the IMCE ontology, together with the terminology currently in use in the APGen DSL. Note that the correspondence established in the table is tentative and by no means rigorous; it captures the current state of an IMCE/APGen bridge which is still under construction. For instance, IMCE does not currently have a formal definition for what constitutes a subsystem, although the word *subsystem* is used informally in the examples mentioned in Ref. [4]. The IMCE ontology uses an abstract term, *behaving element*, to capture the essence of a system component; in fact, a component can be looked at as a concrete implementation of the abstract notion of *behaving element*. On the APGen side, the word *subsystem* is used rather informally; it sometimes describes a component of the system (DSN Station, Solar Panel, Payload), but at other times it represents a point of view rather than the result of decomposing the system into a hierarchy (Geometry, Power). For the time being, we will not worry about these important details and we will adopt the usage commonly found in simulation work. In particular, we will refer to subsystems as if they were components of the system, realizing that reality is often more complex.

Table 3 - Elements of a Generic Subsystem

Element		***Description***	***APGen DSL terminology***
Parameter	constant	"constant" whose value may evolve slowly over the life of the mission	global variable
	global variable[1]	allowed to change in time	global variable
State Variable	numeric	varies continuously (e. g. a physical quantity)	numeric resource
	discrete	transitions from one discrete value to the next (e. g. an instrument mode); usually accompanied by a table of allowed transitions	state resource (lists possible states), abstract resource (implements transitions)
Behavior		pattern of activities or events designed to meet an objective	activity type (high level)

[1] Not part of the IMCE Ontology

Element		Description	APGen DSL terminology
	goal achiever		command (low level)[1]
	timeline generator[2]	built-in behavior, implemented as a modeling algorithm which describes the unfolding in time of a sequence of resource usage events and/or sub-activities; can be imported from an external model via the user-defined library mechanism	modeling section within an activity type or abstract resource definition[3]
Constraint	goal	a constraint imposed on a state over a time interval, usually to achieve a high-level goal	activity type[4]
	state constraint	a passive constraint, which is evaluated during modeling but not enforced	constraint
	goal elaboration[5]	a scheduling constraint, used to determine the placement of desired activities in the mission plan	scheduling section of an activity type definition

In the following sub-sections, we show how the generic description in table 3 applies to concrete subsystems such as those found in the Europa Clipper mission.

1. Deep Space Network (DSN)

The DSN model is mostly mission-independent, since the same ground stations and antennas are used for all deep space missions. The model needs to be extended for missions such as Europa Clipper which rely on assets from space agencies other than NASA, but in that case also, the additional assets do not vary from one mission to the next other than upgrades to the ground system. Table 4 below lists the specifics of the DSN model.

Table 4 - DSN Subsystem

Element	Description	Implementation Details
Parameters	constant, mission-independent	examples: lists of DSN stations, antenna size and other operational characteristics for each station, SPICE kernels for ground station geometry [5]
State Variables	discrete states for station visibility, operational mode etc.; aggregate states expressing global information such as availability of any DSN station for uplink or downlink	
	continuous states for station elevation, telecom link capacity, one-way light time	station elevation and link capacity affect available data rates; one-way light time is of the order of an hour for a spacecraft in orbit around Jupiter
	station allocations	the spacecraft can only use stations that are not only visible but have also been allocated to the mission by the DSN
Behavior	built-in behavior of signal acquisition and other station variables is specified by a detailed multi-mission model that takes all operational constraints into account	state computations are mission-independent (except the simulation of station allocations, if required); computations require assistance from Geometry and Telecom subsystems
Constraints	DSN-related goals	standard activity types are available for implementing all DSN-related requests (uplink and downlink events)
	scheduling constraints may be needed to simulate the allocation of DSN stations to the mission in pre-Phase A, before DSN schedules have become available	

[1] APGen models commands as activity types, but can create special output products (e. g. sequence files) specifically for activities that represent commands

[2] The IMCE Ontology includes timelines but not the means to generate them (other than the equations of physics)

[3] If the model is implemented externally, the modeling section relies on the API specified in the user-defined library to express the required behavior

[4] An activity type that implements a goal should contain decomposition or modeling information that enforces the constraint imposed by the goal

[5] Elaboration is in the process of being added to the IMCE Ontology

2. Guidance, Navigation and Control (GNC)

Table 5 below highlights the components of the GNC subsystem. Although GNC implementation varies from one mission to the next, computational components of the model such as quaternion algebra and matrix manipulation are mission-independent, providing significant heritage and reuse opportunities. High-level behaviors as encoded e. g. by maneuver activity types can be at least partially table-driven, e. g. in the specification of generic turn profiles when changing pointing attitudes.

Table 5 - Guidance, Navigation and Control (GNC) Subsystem

Element	***Description***	***Implementation Details***
Parameters	constants that parameterize the spacecraft hardware: star trackers, inertial measuring units, sun sensors, reaction wheels, thrusters	data structure types are essentially mission-independent but values may vary from one mission to the next
	constants that capture the pairing of mechanical and electronic devices	note: most devices combine a mechanical component (e. g. a reaction wheel) with one or more control electronics circuit board(s)
	constants that capture spacecraft and payload geometry	SPICE Frames are used to specify mounting geometry of imaging and radar instruments
State Variables	numeric states representing position and velocity data obtained or computed from planetary and target body ephemerides	GNC computations need heavy assistance from Geometry and Power subsystems
	numeric states describing spacecraft attitude, encoded as quaternions	
	discrete states describing high-level pointing modes (e. g. Earth- or Nadir-pointed)	
Behavior	Trajectory Correction Maneuvers (TCM) - goal is to achieve a given delta V	activity parameters are provided by the Navigation Team
	turn activities - goal is to achieve a given attitude specified symbolically or via a pointing vector or attitude quaternion	requires turn profile computation and multi-level activity decomposition
	low-level ACS commands - goal is to put ACS in a given state	details vary from mission to mission
Constraints	passive constraints related to hard stop avoidance for gimbaled devices (antennas, solar panels)	turn activity algorithms are designed to avoid running into hard stops; passive constraints provide algorithm validation
	passive constraints related to Sun and bright-body avoidance	require Field of View (FOV) data as well as mounting geometry for imaging instruments

3. Data Subsystem

The Data subsystem is largely mission-independent thanks to its (mostly) table-driven design. Tables need to be provided for

- List of APID's (APplication IDentification), used to tag chunks of data so as to specify their provenance and priority level
- Spectral bands used by transmitters and receivers (e. g. Ka-band, X-band)
- Data rates for production of engineering data by the spacecraft and each instrument
- Priority tables for determining the order in which data is downlinked

Activities that achieve data-related goals such as downlink, uplink, delete and re-transmit are largely mission-independent, with minor modifications that are hand-coded by adapters. Table 6 below summarizes the elements of the Data subsystem.

Table 6 - Data Subsystem

Element	***Description***	***Implementation Details***
Parameters	tables for APID's, data rates, priority levels	table structure is mission-independent; table entries vary from mission to mission
State Variables	continuous states include data rate and data volume states which are defined for each data-producing or data-consuming device onboard the spacecraft	behavior of rate/volume pairs of states is mission independent; parameter tables provide most of the mission specifics
	discrete states based on APID tables	

Element	***Description***	***Implementation Details***
	discrete states based on spectral band (Ka-band, X-band)	
Behavior	standard goals: downlink, uplink, retransmit, delete	
	data transfer goals - move data from one storage device to another, from a storage device to a transmitter, or from a receiver to a storage device	
	budget-setting goals are used e. g. in tactical operations when activity plans need to be refined into specific science observations	
Constraint	scheduling constraints are used to create downlink activities based on availability of both high-priority data and DSN receiving stations	
	passive constraints monitor the filling level of onboard storage devices	

4. Power Subsystem

In a number of recent APGen adaptations, the power subsystem is implemented externally, using the high-fidelity MMPAT power simulation library developed by Eric Wood and his collaborators at JPL [6]. The user-defined library mechanism is used to link the library to the APGen simulation engine. The MMPAT API makes it possible for APGen to ingest tables that describe the Power Equipment List (PEL) as well as detailed load information for each possible state of the devices listed in the PEL.
When an MMPAT-based model is not available, the power subsystem is implemented at a lower level of fidelity, using generic algorithms for estimating the most important state variables of the power model:

- device-specific power consumption based on device-indexed load tables
- battery State of Charge (SOC) based on a simple battery model
- solar power or RTG power output based on a simple solar cell model coupled with attitude and occultation data from the Geometry subsystem

Table 7 below summarizes the contents of the power subsystem.

Table 7 - Power Subsystem

Element	***Description***	***Implementation Details***
Parameters	Power Equipment List (PEL)	imported from the MMPAT model (when used); imported from the System Model (Europa Clipper)
State Variables	PEL states	
	PEL transitions	
	numeric states to capture simulation output (e. g. battery SOC, total load, device-specific dissipation, accumulated energy)	
Behavior	goals are not normally part of the power subsystem; instead, timeline-generating elements are inserted into the behavioral model of every power-consuming or power-generating device onboard the spacecraft	timeline-generating elements are implemented as calls to abstract resources in APGen; when the MMPAT library is available, these calls delegate the modeling to MMPAT via its API
Constraints	passive constraints monitor the battery SOC, which is usually not allowed to fall between a given percentage level	
	additional constraints can be added as needed, e. g. to verify that science observations are staying within their energy allocation	

5. Geometry Subsystem

The Geometry subsystem of most APGen adaptations combines four basic elements:

1. the SPICE toolkit from NASA's NAIF [5] and associated data (called “kernels”)
2. interfacing routines which expose a small fraction of the SPICE API to the APGen DSL
3. utility functions coded in the APGen DSL for performing elementary tasks such as matrix multiplication or vector normalization
4. a number of geometry-based state variables which, although mission-specific, share a common structure

The ability to share a common structure is largely due to the design of the SPICE toolkit, which provides uniform and easy access to geometric information relative to planetary and other target bodies as well as spacecraft for all NASA missions. Of all subsystems, the Geometry subsystem is the one that has the highest level of interaction and integration with other subsystems. As we noted at the beginning of section IV, subsystems often represent specialized view of the entire system, as opposed to a piece of a higher-level aggregate. That is certainly the case for the Geometry subsystem. In spite of its unique characteristics, we have done our best to capture the elements of the Geometry subsystem in the same style as other systems in Table 8 below.

Table 8 - Geometry Subsystem

Element	***Description***	***Implementation Details***
Parameters	constants that encode mission-specific SPICE parameters, such as spacecraft ID, planetary body ID, target body ID	
	tables of Trajectory Correction Maneuvers (TCMs) and Orbit Transfer Maneuvers (OTMs)	maneuver timing and delta V information is provided by the navigation team
State Variables	a variety of numeric states are extracted from planetary ephemerides and SPICE trajectory data (SPK files)	
	GNC quaternion states for attitude-related computations could just as well have been implemented here	quaternion states are shared with the GNC subsystem
	discrete states used to represent geometric conditions of interest to other subsystems (eclipses, occultations, DSN station visibility etc.)	
Behavior	the geometric subsystem does not normally include goals, except for *in situ* missions such as MER and MSL [7]	"geometric goals" such as driving to a target or deploying a robot arm require assistance from 3D-capable applications
Constraints	scheduling constraints are often used for the automatic placement of remote-sensing observations, e. g. whenever the spacecraft is within a specified distance of the target body	
	passive constraints are used to ensure the safety of many instruments (e. g., Sun and bright-body avoidance) and for various engineering purposes (e. g., make sure that enough bright stars are within the FOV of a star tracker)	

A final note on the Geometry subsystem: more than any other, this subsystem benefits from visualization tools that make the system's geometry and its relationship to its planetary environment intuitively obvious. The geometric aspects of APGen simulations have benefited enormously from the integration of APGen with NAIF-maintained tools such as Cosmographia [8], [9]. In fact, a number of problems with spacecraft design and launch geometry were first identified by looking at Cosmographia movies based on APGen simulation output.

6. Telecom Subsystem

The Telecommunications subsystem, although more specialized than the Geometry subsystem, is similar to it in the sense that it represents a point of view rather than a piece of the system. Telecommunications usually take place between a spacecraft and an Earth station, although Mars rovers also communicate with Mars orbiters via relay operations. Any state variable or behavioral specification dealing with telecommunications will therefore make reference, explicitly or implicitly, to the DSN, geometry and data subsystems. Table 9 below describes the specifics of the Telecom subsystem.

Table 9 - Telecom Subsystem

Element	***Description***	***Implementation Details***
Parameters	constants that describe antenna number, antenna gain data tables, transmitter power, available data rates and other operational characteristics of the onboard system	a multi-mission, matlab-based version of JPL's Telecom Forecast Predictor (TFP) model is becoming available as a library suitable for linking to APGen; until now, *ad hoc* models have been linked via the user-defined library or coded in the APGen DSL
	DSN antenna characteristics are mission-independent and available as external models	
State Variables	numeric states such as signal-to-noise ratio	
	discrete states such as maximum sustainable data rate	the computation of windows for telecom opportunities involve power and geometric subsystems as well as the telecom subsystem itself

Element	*Description*	*Implementation Details*
Behavior	the Telecom subsystem does not normally provide goals	
	state computations are based on the physics of the radio wave transmission process	
Constraints	scheduling constraints for telecom activities (uplink, downlink) usually reside in other subsystems but rely on rate estimates provided by the telecom subsystem	

7. *Propulsion Subsystem*

The propulsion subsystem plays an essential role in modeling maneuvers (TCMs, OTMs) and thruster-based turn or slew activities. It is usually mission specific, although early, notional versions of the propulsion subsystem can be put in place for simulations of a project in Phase A or pre-Phase A. Table 10 below lists the specifics of the subsystem.

Table 10 - Propulsion Subsystem

Element	*Description*	*Implementation Details*
Parameters	thruster number and operational characteristics	when possible, propulsion models are imported as libraries provided by the spacecraft team; such libraries can be reused for providing missing detail when spacecraft design has not been finalized
	fuel composition, tank capacity	
	fuel consumption tables	
State Variables	numeric states are used to track fuel consumption and thruster firing count	
	in typical adaptations, delta V information has been processed by the navigation team at the time the spacecraft trajectory was designed; it does not need to be looked at in mission simulations	
Behavior	maneuvers, whose goal is to achieve a specified delta V	require a number of sub-activities created via decomposition algorithms designed to avoid triggering constraint violations
	turns, whose goal is to achieve a specified attitude	similar in structure to maneuvers
	commands, for achieving low-level goals such as thruster prep, turning on heaters etc.	notional until actual commands become available from the spacecraft design team
Constraints	passive constraints can keep track of the number of times each thruster has been fired	

8. *Payload Subsystem*

The bulk of the Payload subsystem consists of instrument-specific activities such as science observations, calibrations and other maintenance activities. There is considerable overlap between instrument parameters and state variables and those stored in the geometry, GNC, power and data subsystems, since most instruments affect many of the state variables on all of these subsystems as well as their own. There can be considerable variation in the details of an instrument-specific adaptation; table 11 below is a generic sketch of what these adaptations may contain.

Table 11 - Payload Subsystem

Element	*Description*
Parameters	typically, instrument parameters include constant lists and maps which • provide operational mode designations • relate power consumption and data rate to operational mode • are used in expressing pointing and other constraints • define mounting and field of view geometry • capture the capacity of internal storage devices • state durations of key instrument observation and maintenance activities
State Variables	a number of discrete states are typically used to describe the operational mode of each instrument
	specific geometry-related state variables may be defined within the payload subsystem, which is more efficient that recomputing these states within the geometry subsystem
Behavior	science observations, which result from science data acquisition goals, are usually implemented as activities that are added to the plan via instrument-specific scheduling criteria

Element	*Description*
	a number of instrument-specific, low-level functions are typically provided to facilitate the description of built-in instrument behavior
Constraints	scheduling constraints are generally expressed as Boolean combinations of geometric factors and discrete state variables such as high-level pointing mode
	a number of passive constraints typically express pointing constraints such as Sun and bright-body avoidance

C. Ideal System Qualities Revisited

1. The Subsystem - System Quality Correspondence

Having outlined the nature of simulation subsystems in some detail, we now revisit our brief discussion of the desirable qualities of the ideal simulation system, and in particular the contents of table 1 at the beginning of this section. Table 12 below lists, for each subsystem, the estimated relevance of the various "-ilities" assigned to a simulation system.

Table 12 - Relevance of System "-ilities" by Subsystem

Subsystem	*Need for Tweakability*	*Need for Evolvability*	*Need for Integrability*
DSN	low; the DSN model has been validated by many missions	low; the DSN changes slowly with time	moderate; it could be useful to export the DSN subsystem to external simulations
GNC	low; the GNC model tends to be generic and few fixes are required	high; a non-standard spacecraft ACS may require mimicking or even importing portions of flight software code[1], which may not be available until Phase C	high; in view of its complexity, it is important for the GNC subsystem - whether imported from flight software or not - to work in harmony with other subsystems
Data	high; instrument data-related behavior may not be fully specified until Phase C	high; instrument data-related behavior may not be fully specified until Phase C	high; the system model will improve in fidelity if it integrates instrument models developed by the instrument teams themselves, when available
Power	high; actual spacecraft and payload behavior always changes after launch, when the spacecraft enters its true environment	moderate; most changes in the power subsystem occur in pre-Phase A and Phase A, when major tradeoffs are investigated	high; virtually everything onboard the spacecraft needs or provides power
Geometry	moderate; pre-Phase A and Phase A tradeoff studies often involve re-positioning instruments or other spacecraft components	low; most geometry calculations are generic and do not change with time	high; the orientation and mounting point of everything attached to the spacecraft needs to be known at some point
Telecom	low; the operational characteristics and link analysis computations of the telecom system do not change much over time	low; telecom subsystem design does not normally change over time	moderate; telecom analysis is relevant to the mission strategy regarding downlink scheduling and science data budget allocations
Propulsion	low; details of the propulsion system have usually been taken into account by the navigation team and do not have a major impact on the rest of the simulation	low	low

[1] As an example, a SEQGEN based verification model for the Dawn mission was successfully modified to include parts of the ACS flight code, resulting in much increased fidelity in predicting turn durations.

Subsystem	***Need for Tweakability***	***Need for Evolvability***	***Need for Integrability***
Payload	high; adjusting the model to improved understanding of instrument behavior is a priority for Phases A through D	high; adjusting the model to improved understanding of instrument behavior is a priority for Phases A through D	high; the system model will improve in fidelity if it integrates instrument models developed by the instrument teams themselves, when available

Note the following two observations about the table:

1. We have omitted "verifiability" from the table, because model validation is always essential in a system model. More precisely, the extent to which the system model can be validated becomes more and more important as the project evolves through successive Mission Phases. More on this topic below.
2. We interpret "tweakability" as the ability to change the model as a result of an unforeseen event, for example discovery of a bug or the emergence of a new requirement; "evolvability" is seen as the ability to go along with planned changes in mission subsystems as they evolve over the life of the mission.

2. Validation Issues

The requirements on the fidelity and accuracy of a system model evolve as the mission evolves through its successive Phases:

- Throughout the life of the system model, adapters are performing checks on their contributions to the overall modeling code. For example, the initial implementation of the Geometry subsystem and subsequent changes to it are checked using the NAIF Web Geo Calc facility [9].
- In pre-Phase A and in Phase A, one of the main functions of the system model is to enable the study of tradeoffs between design options. The main requirement on fidelity is that the relative performance of the various options being considered should be modeled as accurately as possible. Validation efforts should focus on the model's accuracy in estimating relative performance of the components under study.
- In Phase C, detailed specifications for spacecraft and instrument commands are becoming available to the modeling team. It may make sense at that point to integrate an external model of the onboard Command and Data Handling (C&DH) subsystem into the system framework, if such a model is available.
- In Phase D, actual behavior of the assembled spacecraft and payload assembly is being measured for the first time. This is when subsystem tweakability comes into play in a major way, especially for the payload subsystem whose instrument models may never have been validated before. This is also a time at which it becomes important for the system model to provide predicted telemetry data, so as to make comparison with actual behavior as convenient as possible.
- Post-launch behavior always results in further changes to the system's behavior, once the spacecraft has entered its true environment.
- Phase E usually extends over a long period of time, and it is fairly common for spacecraft components to exhibit off-nominal behavior as they wear out or encounter unforeseen circumstances. Subsystem models have to be modified accordingly, and obtaining agreement between predicted vs. actual telemetry becomes a critical goal of the validation effort.

V. Why Integration Efforts Fail

In the previous section, we have attempted to paint an admittedly sketchy but nonetheless realistic picture of the many elements and, more importantly, the many kinds of elements that enter the design and implementation of the subsystems that make up a complete mission simulation framework. In order to streamline the information we presented, we have taken hints from the JPL IMCE initiative in an attempt to use uniform terminology across all subsystems. Now, we want to take an even bigger step towards uniformity and generality and seek overall estimates of the global effort required to put together a simulation infrastructure similar to, but hopefully better designed than, the APGen models that have provided us with basic modeling data. We will argue that the chances of success of any such effort, assuming zero probability for miracles and extraterrestrial interference, are of the order of 5% in the best of cases.

To start, consider the ISO-OSI model known as the "Open System Interconnection" layers; we have reproduced it in Table 13 below.

Table 13 - ISO-OSI Open System Interconnection Layers

Realm	***Layer Name***
Application	7. Application
	6. Presentation
	5. Session
Data Flow	4. Transport
	3. Network
	2. Data Link
	1. Physical

This model captures in detail all the things that have to be present for a software package (the Application) to connect in a fully functional manner to a network. Although this model is not quite appropriate for our purposes - for example, linking subsystems in our case does not necessarily require network access - we can use it as inspiration to put together an "integration layer diagram" which captures the various elements that contribute to subsystem integration. Needless to say, we will also seek inspiration from the actual subsystem descriptions laid out in the previous section. Our first attempt is illustrated in Table 14 below (we adopt the "reverse numbering system" of ISO-OSI for consistency).

Table 14 - Steps Involved in Integrating an External Subsystem into the APGen Infrastructure

Realm	***Step Name or Description***	
Modeling Engine	7. synchronization with other subsystems	
	6. setting / querying remote state variables	
	5. opening and closing a modeling run	
User-defined Library	4. API for configuration management	find configuration file(s)
		load configuration(s)
		unload configuration(s)
	3. API for opening and closing a simulation session	
	2. wrap the external API into an API suitable for the user-defined library	
External Model	1. design and implement the external model as a library or server	

The above table assumes that the external subsystem has already been configured to be called from a separate entity such as the APGen modeling engine. But what if that is not the case? In our experience, mission engineers and multi-mission tool developers often design simulation tools as standalone applications meant for their personal use or for the use of their colleagues on an individual basis, and it can take a fair amount of work to split such applications into a main program and a library with an API generic enough to be exported outside of the application's realm. We cannot possibly anticipate all the challenges that would have to be met in such situations, but we can certainly explore what it would take to export the APGen simulation engine to another realm. In fact, such an export capability could be very useful in a first step towards distributed processing of independent or nearly independent subsystem models, a topic that deserves attention on its own. But here, let us just concentrate on the steps required for exporting APGen simulation data and processes. Table 15 below summarizes the necessary steps as we see them; this time we number steps from top to bottom for reasons that will become clear shortly.

Table 15 - Steps Involved in Exporting an APGen Model

Realm	***Step Name or Description***	
APGen Modeling Engine	1-3. design and implement an API for exporting internal APGen data	1. export parameters
		2. export state variables
		3. export behaviors that implement high-level goals
APGen Scripting Language and Execution Engine	4. design and implement an API for letting an external entity take control of the APGen engine (e. g., modeling commands)	
	5. design and implement an API for reading plan and adaptation files into APGen	
Modeling Server	6. use the API to attach the APGen engine to a server that implements a chosen Internet access protocol (e. g. HTTP/ReST, SOAP, XmlRPC, ...)	
	7. link the APGen export API to methods or requests that can be submitted to the server by a remote client	

We are now ready to compute our estimated probability of success. The complete integration process, starting with two separate modeling entities that were not designed to work together, and ending with an integrated modeling system that can be operated as a single entity, involves 14 separate steps: 7 on the server side (the model being exported) and 7 on the client side (the system being extended). In fact, if we change the numeric labels in Table 14 so they read 8 to 14 instead of 1 to 7, our two tables provide a complete sequence of steps for integrating two distinct modeling systems. Assuming a highly talented and motivated team, we assign a 90% probability of success to each individual step. Assuming that all 14 steps are independent of each other, we conclude that

$$\textit{Probability of success of the overall integration task} = (0.90)^{14}=0.0523$$

or about 5%, as claimed.

It goes without saying that our assumptions are simplistic and the reader will have no trouble punching holes through most of them. If we absolutely had to come up with a more realistic estimate, how would we go about it? Let us address the two main aspects of the probability calculation:

1. We assumed that the integration steps are independent of each other, which leads to the high exponent (14) in the probability formula. Are there factors that would make the steps more coherent, thereby reducing the magnitude of the exponent?
2. We assumed a uniform 0.90 probability of success for individual steps. Can we do any better than using a wild guess?

Although we cannot come up with quantitative answers, it is easy to identify project characteristics that could affect each one of these two issues. Table 16 below lists two project characteristics that are "centrifugal," i. e., tend to lead the project away from a high level of integration, and two project characteristics that are "centripetal," i. e., tend to drive the project towards tighter integration.

Table 16 - Centrifugal vs. Centripetal Project Characteristics

Centrifugal characteristics	1. organizations do not spontaneously cooperate with other organizations and tend to protect their independence - think of how hard it is to build a consensus among groups of engineers from different departments 2. talented individuals often have a preference for their own solutions over approaches that require cooperation with an outside group
Centripetal characteristics	1. project management can state and enforce a modeling methodology that emphasizes system-level integration from the beginning, thus turning model integration into a "corporate goal" and empowering integration efforts 2. project personnel exposed to the benefits of system-wide integration from the very beginning see the concrete advantages of integration, which encourages participation in integration with no need for prodding from anyone

In an organization that has no way to offset centrifugal factors, we believe that our estimate of 90% probability of success for an individual integration step is highly optimistic. As a result, an effort to build an integrated, mission-wide modeling system will greatly improve its chances of success if it includes one of the two centripetal characteristics listed above, and preferably both. In particular, we note that the availability of detailed simulations from the very beginning in pre-Phase A of the Europa Clipper project was instrumental in enlisting the cooperation of all subsystems to the overall effort. Particularly noticeable was the case of the GNC team, who shared their flight system algorithms with APGen adapters and, in exchange, acquired detailed and realistic attitude-related scenarios which normally take a long time to put together. Equally noticeable was the enthusiastic participation of the instrument teams, who eagerly contributed more and more details about the scheduling process for science observations when they realized that the scenarios coming out of the simulation could help them assess whether and how well the project's scientific objectives could be met.

VI. Towards a Success-Oriented Simulation Architecture

We now build on the system and subsystem considerations of the previous sections to try and answer the two questions listed in the Introduction.

A. Extending the Europa Clipper Framework

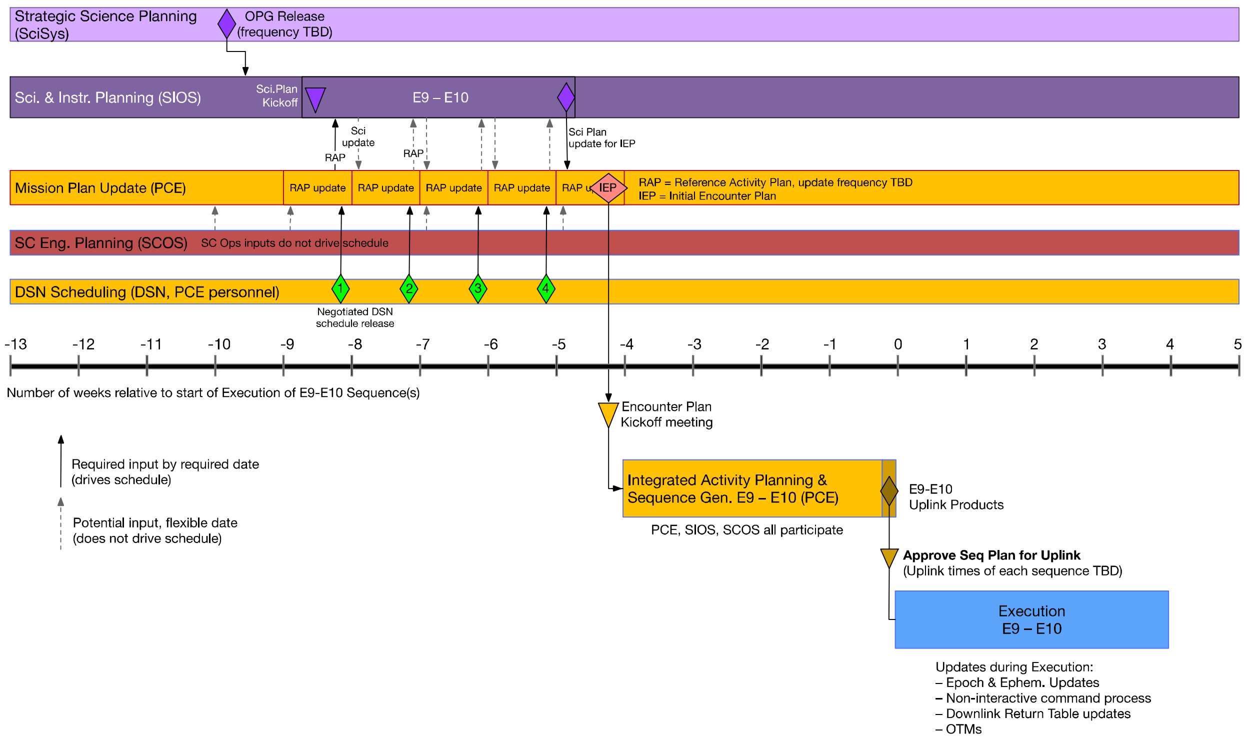
The key challenge in extending the current simulation framework of the Europa Clipper mission, currently in Phase B, is to meet the challenges of Phase E, i. e., mission operations.

1. Variable Planning Horizon

While many Phase A simulations were carried out over the entire lifetime of the mission, the planning horizon in Phase E will be much shorter: a single orbit around Jupiter takes about two weeks, and a typical planning horizon will encompass two of those. The proposed planning process for Europa Clipper is outlined in Fig. 1 below. The process illustrated in the figure can be summarized as follows:[1]

- At all times during Phase E, a valid "Reference Activity Plan" (RAP) will be available from the system model. The RAP takes the place of a static document called "strategic plan" used in past missions; unlike the strategic plan, the RAP is a dynamic instance of the system model and can be updated at any time.
- The main outcome of a Phase E planning session will be a RAP update concentrated on a 4-week chunk[2] of the activity plan, covering about two orbits around Jupiter. Part of the RAP update will consist in running a full simulation from that point forward, to make sure that the plan update does not conflict with future planned activities.

Figure 1 - Europa Planning Process (courtesy of D. Bindschadler and the Europa Clipper Project)[3]


Strategic Science Planning (SciSys)
OPG Release (frequency TBD)
Sci. & Instr. Planning (SIOS)
Sci.Plan Kickoff
E9 – E10
Sci update
RAP
RAP
Sci Plan update for IEP
Mission Plan Update (PCE)
RAP update
RAP update
RAP update
RAP update
RAP u
IEP
RAP = Reference Activity Plan, update frequency TBD
IEP = Initial Encounter Plan
SC Eng. Planning (SCOS)
SC Ops inputs do not drive schedule
DSN Scheduling (DSN, PCE personnel)
1
2
3
4
Negotiated DSN schedule release
-13 -12 -11 -10 -9 -8 -7 -6 -5 -4 -3 -2 -1 0 1 2 3 4 5
Number of weeks relative to start of Execution of E9-E10 Sequence(s)
Encounter Plan Kickoff meeting
Required input by required date (drives schedule)
Potential input, flexible date (does not drive schedule)
Integrated Activity Planning & Sequence Gen. E9 – E10 (PCE)
E9-E10 Uplink Products
PCE, SIOS, SCOS all participate
Approve Seq Plan for Uplink
(Uplink times of each sequence TBD)
Execution E9 – E10
Updates during Execution:
– Epoch & Ephem. Updates
– Non-interactive command process
– Downlink Return Table updates
– OTMs

[1] Special thanks to Dave Mohr and Duane Bindschadler for sharing this information

[2] Durations are approximate since Europa flybys are not always regularly spaced; see Ref. [1] for details

[3] This figure is adapted from an architectural diagram presented at the Preliminary Design phase; the use of E9 and E10 is simply as a possible example of approximate timing

2. Web-based Access to the System Model

In Phase A, the APGen model was run by a small team of experts intimately familiar with the configuration and operation of the modeling engine and of the subsystem models attached to it. In Phase E, planning operations will require providing mission engineers with the ability to make changes to key aspects of the model (e. g., scheduling conditions) and to build a modified plan reflecting the changes. This will require significant changes to the simulation infrastructure:

- Web-based access will have to be provided, which requires setting up the simulation engine as a server or pool of servers capable of supporting a team of users.
- The web interface will have to provide users with the ability to modify the adaptation on the fly, such as the details of scheduling constraints for science activities.
- The system model will have to provide easy-to-query archiving storage, allowing planning personnel to store their plans in association with the changes they made to scheduling constraints.

3. Increased Fidelity Requirements

While Phase A simulations can get away with notional commands that just set various state variables to desired values, any activity in the plan is ultimately executed onboard the spacecraft through a sequence of commands. Validation of plan activities therefore requires a detailed simulation of spacecraft behavior while executing those sequences. While such simulations have been successfully carried out using APGen [3], alternatives such as using a specialized C&DH model (possibly adapted from the flight software) should be investigated in the interest of maximizing fidelity and reducing the chances of modeling errors.

It is also conceivable that improved instrument models will become available as science team refine their understanding of their instrument's behavior under the actual, deep-space operating environment. On the other hand, instrument teams have been exposed to the APGen adaptation since pre-Phase A and may have found that the APGen DSL is expressive enough for the purpose of modeling their instrument. In any case, it will help to provide

- a streamlined integration process for the Europa Clipper configuration of the APGen model
- an IDE for the APGen DSL similar to those currently available for established languages such as C++ and Java
- a model debugger which allows APGen adapters to zero in on causes of unexpected behavior without having to become an expert in APGen internals

B. Extending the Europa Clipper Modeling Infrastructure to Future Missions

The Europa Clipper spacecraft is not planned to reach Jupiter until 2025, and so the recommendations we just made about meeting the requirements of Europa Clipper mission operations could be repeated verbatim for any future mission planning to enter operations in the same time frame. The only difference between the Europa Clipper context and the context of a future mission is what we might call the “simulation bootstrapping process,” i. e., the centripetal characteristics which we discussed at the end of section V. On Europa Clipper, these characteristics were largely the result of serendipity - one of us (S. W.) was on the Project Systems Engineering team from the very beginning. For other missions, our recommendation is to rely less on chance and more on the process used in the initial configuration and staffing of a space mission to maximize the centripetal characteristics outlined at the end of section V.

The results achieved by the Europa Clipper simulation team have already captured considerable attention on the part of JPL systems engineers, and we will rely on the collective wisdom of the engineering community for making system-wide simulations a basic requirement of any new space mission setup. Our description of the Europa Clipper system - the APGen modeling engine and the subsystem models attached to it - will hopefully tell systems engineers what they need to know to put together a system as capable as or more capable than the one used on Europa Clipper.

VII. Conclusion

In our paper, we have provided the reader with a whirlwind tour of the current Europa Clipper simulation system, what makes it tick, and what it took to assemble the system from scattered pieces. We believe that the success of this simulation system speaks for itself and that future missions will have a strong incentive to replicate it in some form. We also acknowledge that assembling such a system takes a significant combination of talent, know-how and dedication which may not be so easily found on other projects. The bar of entry into the world of mission-wide simulation systems could be lowered quite a bit by implementing the recommendations made earlier:

- turn the simulation engine into a modeling server or pool of servers
- provide web access to the simulation engine and to key aspects of the behavioral and constraint models

- train mission personnel in the art of subsystem integration by teaching the methodology outlined in section V
- last but not least, strive to adopt the centripetal characteristics listed at the end of section V

We are confident that these recommendations will result in simulation systems that will prove as capable as the one we helped develop for the Europa Clipper mission.

Acknowledgments

The research was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration. The authors wish to extend special thanks to Duane Bindschadler, Eric Ferguson, Dave Mohr and Dennis Page for useful comments and discussions about the subject matter of this paper.

References

[1] Ferguson, E. W., Wissler, S. S., Bradley, B. K., Maldague P. F., Ludwinsky, J. M., and Lawler, C. M., “Improving Spacecraft Design and Operability for Europa Clipper through High-Fidelity Mission-Level Modeling and Simulation,” *AIAA 15th International Conference on Space Operations (SpaceOps)*, Marseille, France, 2018 (to be published).

[2] Maldague, P.F., Wissler, S., Lenda, M., and Finnerty, D., “APGEN Scheduling: 15 years of Experience in Planning Automation,” *AIAA 13th International Conference on Space Operations (SpaceOps)*, Pasadena, CA, 5-9 May 2014. doi: 10.2514/6.2014-1809

[3] Wissler, S., Maldague, P. F., Rocca, J., and Seybold, C., “Deep Impact Sequence Planning Using Multi-Mission Adaptable Planning Tools with Integrated Spacecraft Models,” *AIAA 9th International Conference on Space Operations (SpaceOps)*, Rome, Italy, 2006. doi: 10.2514/6.2006-5869

[4] Castet, J.-F., Rozek, M. L., Ingham, M. D., Rouquette, N. F., Chung, S. H., Kerzhner, A. A., Donahue, K. M., Jenkins, J. S., Wagner, D. A., Dvorak, D. L., Karban, R., “Ontology and Modeling Patterns for State-Based Behavior Representation,” *2015 AIAA Modeling and Simulation Technologies Conference (SciTech Forum),* Kissemmee, FL, 5-9 Jan. 2015. doi: 10.2514/6.2015-1115

[5] Acton, C.H., “Ancillary Data Services of NASA’s Navigation and Ancillary Information Facility,” *Planetary and Space Science*, Vol. 44, No. 1, pp. 65-70, 1996.

[6] Wood, E., and Adamson, A., “Multi-Mission Power Analysis Tool”, URL: http://www.techbriefs.com/component/content/article/1059-gdm/tech-briefs/9446-npo-47290 [cited Apr. 8, 2014].

[7] Mitchell, A., Bresina, J., et al, “MAPGEN: Mixed-Initiative Planning and Scheduling for the Mars Exploration Rover Mission,” *IEEE Intelligent Systems [online journal],* IEEE1094-7167/04, pp8-12, URL: http://ieeexplore.ieee.org/iel5/9670/28315/01265878.pdf?arnumber=1265878 [cited 10 May 2006].

[8] Acton, C., Bachman N., Semenov, B., Wright, E., “A Look Toward the Future in the Handling of Space Science Mission Geometry,” *Planetary and Space Science*, Vol. 150, Jan. 2018, pp. 9-12. doi: 10.1016/j.pss.2017.02.013

[9] Semenov, B. V., “WebGeocalc and Cosmographia: Modern Tools to Access OPS SPICE Data,” AIAA 15th International Conference on Space Operations (SpaceOps), Marseille, France, 2018 (to be published).